????????

????? ??????

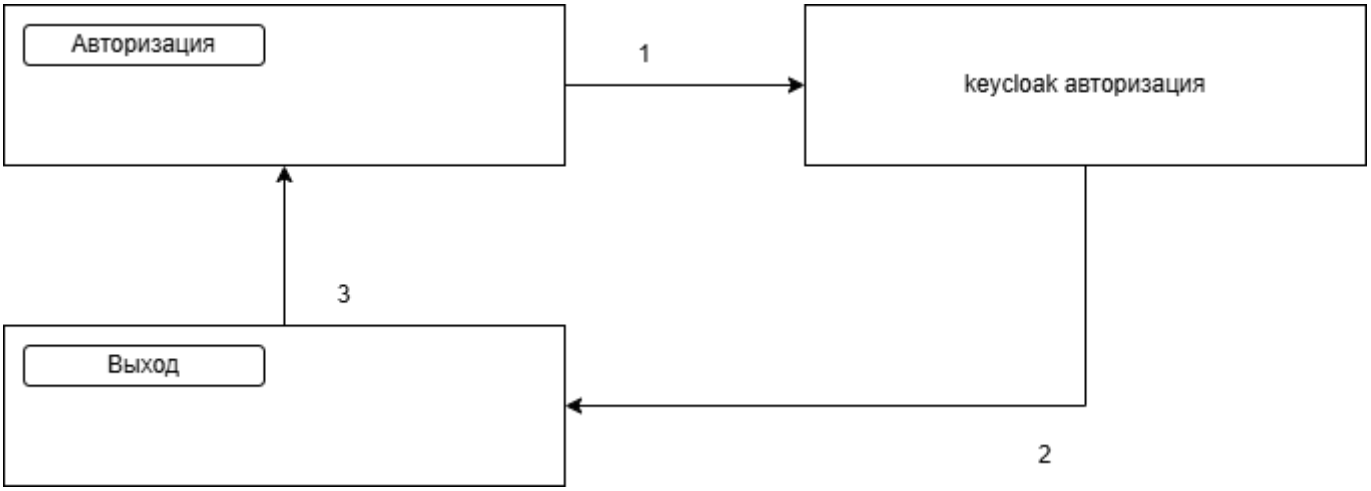
Задача: Необходимо организовать защиту приложения со следующими требованиями:

- Авторизация по логину и паролю
- Разделение пользователей на группы
  - Администратор (все права)
  - Руководитель (просмотр статистики по всем направлениям)
  - Руководитель направления (права в пределах направления, нет возможности просмотра других направлений)
  - Руководитель предприятия (права в пределах предприятия, нет возможности просмотра других объектов)
- Разделение пользователей по ролям в соответствии с группами
- На backend есть API endpoint и HTML endpoint
- Неавторизованный пользователь имеет доступ к некоторым страницам без авторизации

Реализуем это при помощи декораторов из отдельного модуля. Желательно не хардкодить роли, должно быть внешнее хранилище.

??????? ???????????

Реализуем следующий процесс авторизации:



Делаем страницу с проверкой, авторизован или нет пользователь. Если пользователь не авторизован, показываем кнопку Авторизация. Если авторизован - кнопку Выход. Кнопки отличаются ссылками и текстом.

Будем использовать модуль python-keycloak

```
pip install python-keycloak
```

Адресация серверов

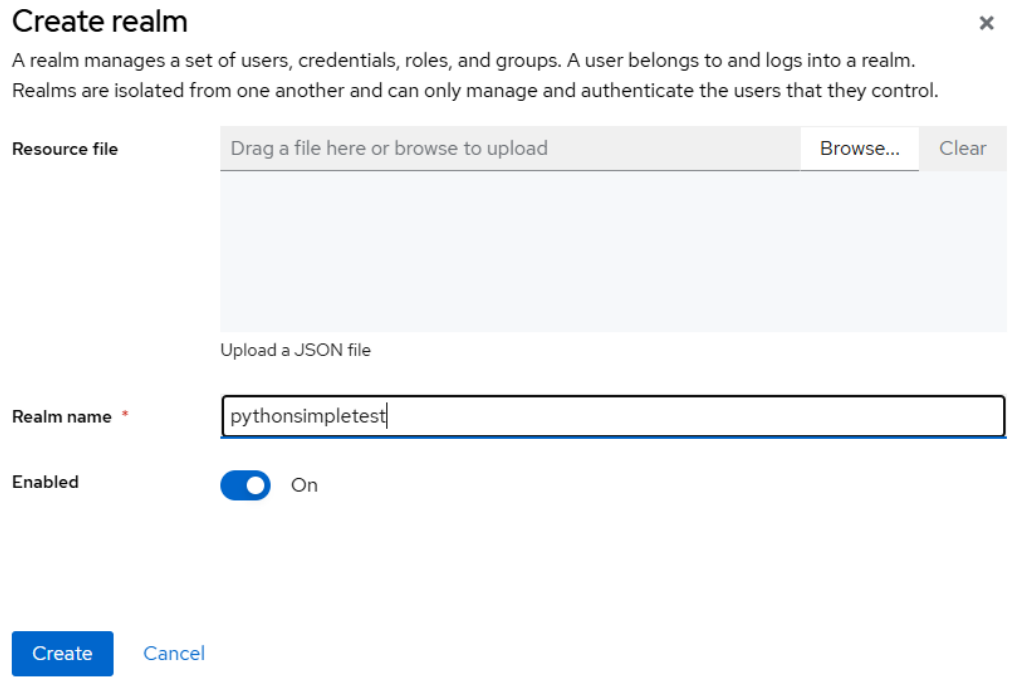
192.168.1.3 web server и ПК, с которого я тестирую работу

192.168.1.195 keycloak server

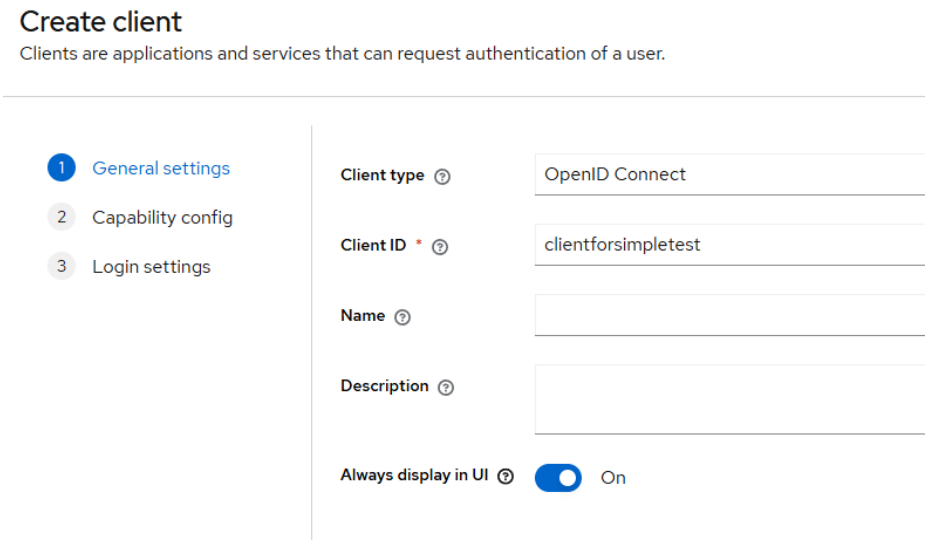
Настройка keycloak.

Создаем realm для данного эксперимента. Назовем его pythonsimpletest.

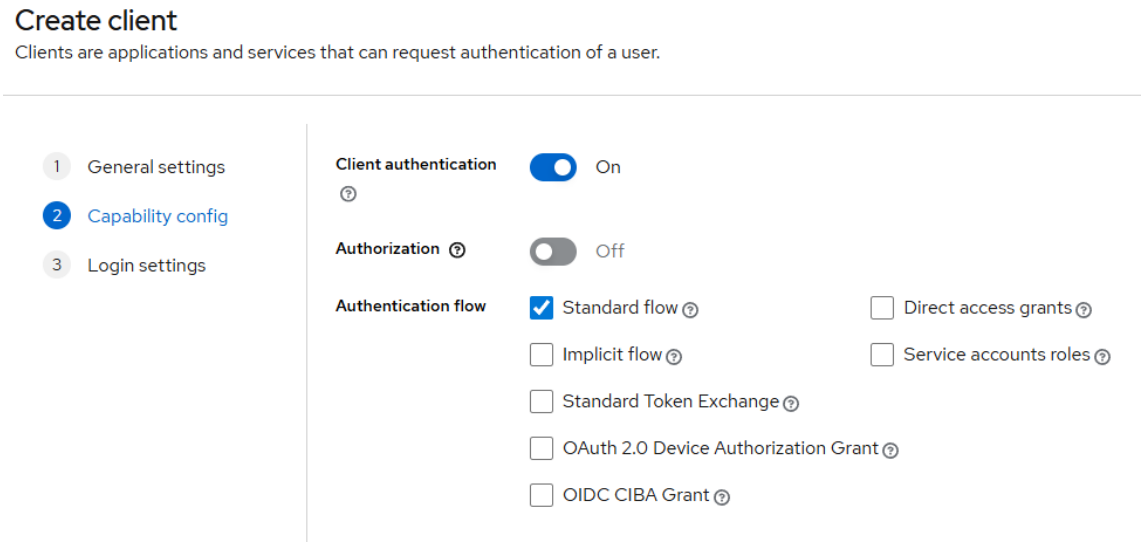
В разделе Manage realms - Create

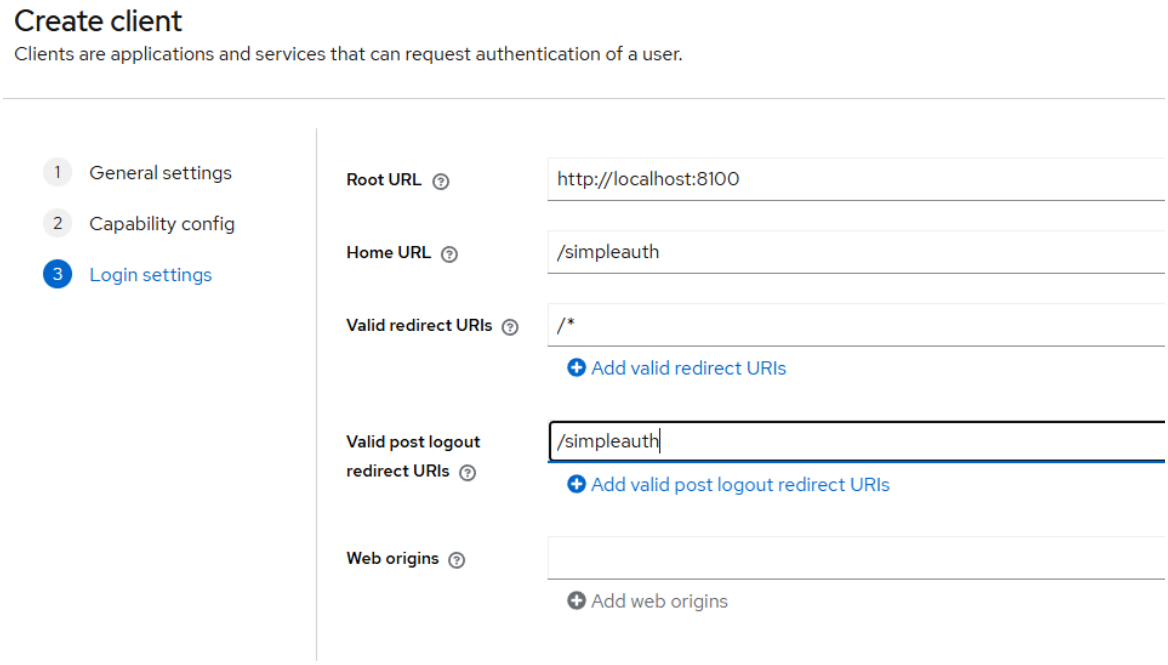


Теперь создаем клиента. Назовем его clientforsimpletest. Clients - Create client

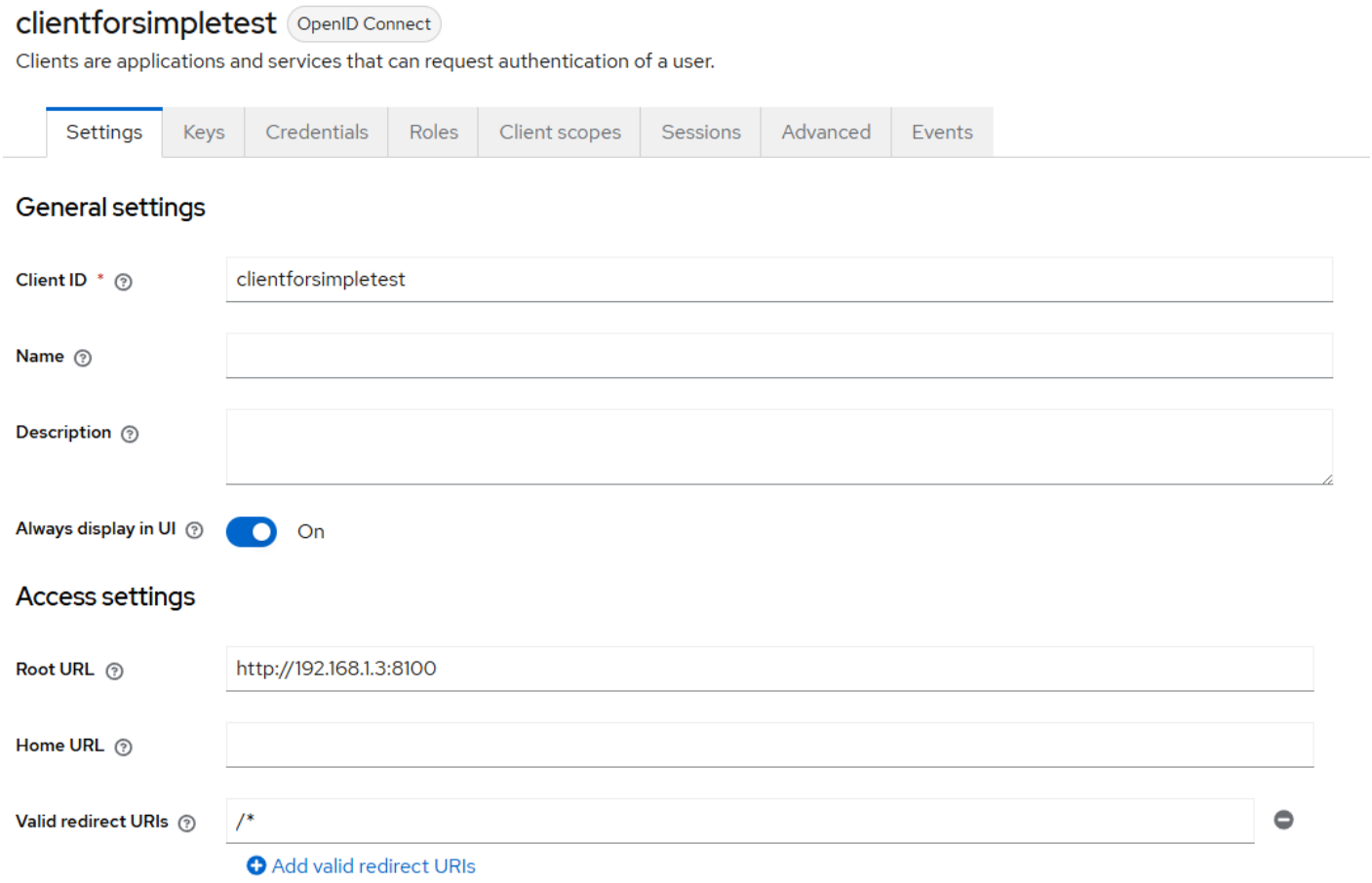


Поскольку этот клиент доверенный и расположен на сервере, то Client authentication включаем,





И тут проявилась первая ошибка. Localhost виден с моего ПК. Поэтому, когда я прописал на сервере keycloak в разделе Root URL localhost - ничего не заработало. Похоже, что необходимо указывать имена/ip доступные с сервера keycloak а не только с браузера на ПК пользователя. Результирующие настройки клиента:



Создаем пользователя и задаем ему пароль.

Python клиент

```
from fastapi import FastAPI, Depends, Request
from fastapi.responses import HTMLResponse, RedirectResponse
from uvicorn import run
from typing import Optional

from keycloak import KeycloakOpenID

app = FastAPI(docs_url=None, redoc_url=None, openapi_url=None)

KEYCLOAK_URL = "http://192.168.1.195:9090/"
REALM_NAME = "pythonsimpletest"
CLIENT_ID = "clientforsimpletest"
CLIENT_SECRET = "Vix6txRyHwt81KyZIpl7O06CxpWMFtib"
REDIRECT_URI = "http://192.168.1.3:8100/auth/callback"

# Инициализация Keycloak клиента
keycloak_openid = KeycloakOpenID(
```

```
    server_url=KEYCLOAK_URL,
    client_id=CLIENT_ID,
    realm_name=REALM_NAME,
    client_secret_key=CLIENT_SECRET,
)


async def get_current_user_from_cookie(request: Request) -> Optional[dict]:
    """Извлечение и валидация пользователя из cookie"""

    # Получаем токен из cookies
    access_token = request.cookies.get("access_token")

    if not access_token:
        return None

    try:
        # Проверяем токен через Keycloak
        userinfo = keycloak_openid.userinfo(access_token)
        return userinfo
    except Exception as e:
        # Если токен просрочен или невалиден, удаляем его
        # Но здесь мы не можем удалить cookie, это нужно делать в ответе
        return None

def get_login_url():
    """Генерация URL для входа через Keycloak"""
    auth_url = keycloak_openid.auth_url(
        redirect_uri=REDIRECT_URI,
        scope="openid email profile",
        state="random_state_string"  # В реальном приложении используйте генерацию случайной
строки
    )
    return auth_url

@app.get("/", response_class=HTMLResponse)
async def simpleauth(request: Request):
    user = await get_current_user_from_cookie(request)
    if not user:
        login_url = get_login_url()
```

```
        html_content = f'<!DOCTYPE html><html><body><a
href="{login_url}"><button>Авторизация</button></a></body></html>'
    else:
        html_content = '<!DOCTYPE html><html><body><a
href="/logout"><button>Выход</button></a></body></html>'
    return HTMLResponse(content=html_content, status_code=200)

@app.get("/auth/callback")
async def auth_callback(code: str):
    """Callback URL для обработки ответа от Keycloak после логина"""
    try:
        # Обмен кода на токены
        token_response = keycloak_openid.token(
            grant_type="authorization_code",
            code=code,
            redirect_uri=REDIRECT_URI
        )

        access_token = token_response.get('access_token')

        # Создаем редирект с токеном в заголовке (через установку cookie)
        response = RedirectResponse(url="/")

        # Устанавливаем токен в cookie (альтернатива Authorization header для браузера)
        response.set_cookie(
            key="access_token",
            value=access_token,
            httponly=True,  # Защита от XSS
            secure=False,   # True для HTTPS
            samesite="lax"
        )
        refresh_token = token_response.get('refresh_token')
        response.set_cookie(
            key="refresh_token",
            value=refresh_token,
            httponly=True,  # Защита от XSS
            secure=False,   # True для HTTPS
            samesite="lax"
        )
        return response
```

```
    except Exception as e:
        return HTMLResponse(content=f"<h1>Ошибка авторизации: {str(e)}</h1>", status_code=400)

@app.get("/logout")
async def logout(request: Request):
    """Выход из системы"""
    refresh_token = request.cookies.get("refresh_token")

    # Логаут через python-keycloak
    if refresh_token:
        try:
            keycloak_openid.logout(refresh_token)
        except Exception as e:
            print(f"Keycloak logout error: {e}")
    response = RedirectResponse(url="/")
    response.delete_cookie("access_token")
    response.delete_cookie("refresh_token")
    return response

if __name__ == '__main__':
    run(app="simple:app", host='0.0.0.0', port=8100, workers=4, log_level='warning')
```

?????????? ?????

Теперь добавим scopes в токен. Даже просто добавление списка ролей оказалось той еще задачей. Во многих инструкциях будет сказано, что настройка маппера в Keycloak Admin Console -> ваш realm → Clients → ваш клиент -> Вкладка Mappers. Но это не так. Пример настройки для получения ролей:

- Перейди в Client Scopes: Твой realm → Clients → выбери своего клиента → вкладка Client scopes → кликни на дедикейтед скоп *-dedicated (например, my-client-dedicated).
- Открой вкладку Mappers.
- Нажми Configure a new mapper и выбери By configuration.
- Выбери тип маппера: В выпадающем списке выбери User Realm Role.
- Вот это ключевой момент: использование предустановленного типа, а не создание своего, гарантирует корректную внутреннюю структуру.
- Проверь настройки (они должны заполниться автоматически):
- Name: Оставь realm roles (или любое другое).
- Token Claim Name: Убедись, что здесь realm_access . Это единственное правильное имя поля.
- Add to userinfo: Убедись, что переключатель установлен в ON.
- Add to access token / Add to ID token: Можно оставить OFF, если роли не нужны в самих JWT токенах.
- Сохрани маппер (кнопка Save).

```
async def get_current_user_roles(request: Request) -> Optional[dict]:
    """Извлечение ролей"""
    access_token = request.cookies.get("access_token")
    if not access_token:
        return []
    try:
        claims = jwt.get_unverified_claims(access_token)
        roles = claims.get("realm_access", {}).get("roles", [])
        return roles
    except Exception:
        return []
...
@app.get("/", response_class=HTMLResponse)
async def simpleauth(request: Request):
    user = await get_current_user_from_cookie(request)
```

```
    if not user:
        login_url = get_login_url()
        html_content = f'<!DOCTYPE html><html><body><a
href="{login_url}"><button>Авторизация</button></a></body></html>'
    else:
        roles = await get_current_user_roles(request)
        html_content = f'''<!DOCTYPE html><html><body><a
href="/logout"><button>Выход</button></a>
        <p>Доступные ключи: {user.keys()}</p>
        <p>Доступные роли: {roles}</p>
        </body></html>'''
    return HTMLResponse(content=html_content, status_code=200)
```

?????????? ???????????

Почему-то для fastapi предпочтительнее использовать механизм Depends. Однако мне проще декораторы.

```
from functools import wraps

from fastapi import FastAPI, Request
from fastapi.responses import HTMLResponse, RedirectResponse
from fastapi.concurrency import run_in_threadpool
from uvicorn import run
from typing import Optional

from keycloak import KeycloakOpenID
from jose import jwt
from jose.exceptions import JWTError, ExpiredSignatureError

app = FastAPI(docs_url=None, redoc_url=None, openapi_url=None)

KEYCLOAK_URL = "http://192.168.1.195:9090/"
REALM_NAME = "pythonsimpletest"
CLIENT_ID = "clientforsimpletest"
CLIENT_SECRET = "Vix6txRyHwt81KyZIpl7O06CxpWMFtib"
REDIRECT_URI = "http://192.168.1.3:8100/auth/callback"

KEYCLOAK_PUBLIC_KEY = """-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAtrnsNk3z21imvfNFDT8teaekNgFhnOSuKcZiPc1Iv+qknDg8X1
zF7KJLtZMrCZSLRJM6T754F8KtNUSj0Ioa3ehYSPEdF4SX7tBdacpeDz0GlKWYFa845/e6H2349I+7EuO1bkHfWz/v6n2m
m+jeZKU0ujqr2boBoPWwkMoKwXNMl/Sac0YMlv1fVHiISyREDG+FordAwYlbLVYCD36ckk0UnAKBc59Q36DMiKSx7JpdvR
0vHIeWb4mQFlRbLdmWkK4ebUe+B/k1/cdd3LHdQ6vc1i45bMcJlrFYocDOGK99Mf8pT8OMPQvTJ8cTe9xSkCAx0l7YiSv+
5hfK1C6DswIDAQAB
-----END PUBLIC KEY-----"""
ALGORITHMS = ["RS256"]

# Инициализация Keycloak клиента
keycloak_openid = KeycloakOpenID(
    server_url=KEYCLOAK_URL,
    client_id=CLIENT_ID,
```

```
    realm_name=REALM_NAME,
    client_secret_key=CLIENT_SECRET,
)

# ============================Декораторы авторизации=======================
def verify_token(token: str):
    return jwt.decode(
        token,
        KEYCLOAK_PUBLIC_KEY,
        algorithms=ALGORITHMS,
        audience="account",
        options={
            "verify_signature": True,
            "verify_aud": False,
            "verify_exp": True,
        }
    )

def get_login_url():
    """Генерация URL для входа через Keycloak"""
    auth_url = keycloak_openid.auth_url(
        redirect_uri=REDIRECT_URI,
        scope="openid email profile",
        state="random_state_string"  # В реальном приложении используйте генерацию случайной
строки
    )
    return auth_url

def session_required(func):
    @wraps(func)
    async def wrapper(*args, **kwargs):
        request: Request = kwargs.get("request")
        if request is None:
            for arg in args:
                if isinstance(arg, Request):
                    request = arg
                    break
        if request is None:
            raise RuntimeError("Request object not found")
```

```
        token = request.cookies.get("access_token")

        if not token:
            return RedirectResponse(get_login_url())

        try:
            claims = verify_token(token)
            request.state.user = claims
        except ExpiredSignatureError:
            response = RedirectResponse(get_login_url())
            response.delete_cookie("access_token")
            response.delete_cookie("refresh_token")
            return response
        except JWTError:
            response = RedirectResponse(get_login_url())
            response.delete_cookie("access_token")
            response.delete_cookie("refresh_token")
            return response
        return await func(*args, **kwargs)
    return wrapper

def html_norole():
    html_content = f'''<!DOCTYPE html><html><body><a href="/"><button>На главную</button></a>
    <p><a href="/logout"><button>Выход</button></a></p>
    <p>У вас нет доступа к этой странице</p>
    </body></html>'''
    return HTMLResponse(content=html_content, status_code=200)

def role_secondrole_required(func):
    @wraps(func)
    async def wrapper(*args, **kwargs):
        request: Request = kwargs.get("request")
        if request is None:
            for arg in args:
                if isinstance(arg, Request):
                    request = arg
                    break
        if request is None:
            raise RuntimeError("Request object not found")
        userinfo = request.state.user
```

```
        roles = userinfo.get("realm_access", {}).get("roles", [])
        if 'secondrole' not in roles:
            return html_norole()
        return await func(*args, **kwargs)
    return wrapper
# ================================================================================

@app.get("/", response_class=HTMLResponse)
@session_required
async def simpleauth(request: Request):
    userinfo = request.state.user #await get_current_user_from_cookie(request)
    roles = userinfo.get("realm_access", {}).get("roles", [])

    html_content = f'''<!DOCTYPE html><html><body><a href="/logout"><button>Выход</button></a>
    <p>Доступные ключи: {userinfo}</p>
    <p>Доступные роли: {roles}</p>
    <p><a href="/secret"><button>Доступ к секрету</button></a></p>
    </body></html>'''
    return HTMLResponse(content=html_content, status_code=200)

@app.get("/secret", response_class=HTMLResponse)
@session_required
@role_secondrole_required
async def roleneeded(request: Request):
    html_content = f'''<!DOCTYPE html><html><body><a href="/logout"><button>Выход</button></a>
    <p><a href="/"><button>На домашнюю страницу</button></a></p>
    </body></html>'''
    return HTMLResponse(content=html_content, status_code=200)

@app.get("/auth/callback")
async def auth_callback(code: str):
    """Callback URL для обработки ответа от Keycloak после логина"""
    def append_cookie(response, key, value):
        response.set_cookie(
            key=key,
            value=value,
            httponly=True,  # Защита от XSS
            secure=False,   # True для HTTPS
            samesite="lax"
        )
```

```
        return response
    try:
        # Обмен кода на токены
        token_response = keycloak_openid.token(
            grant_type="authorization_code",
            code=code,
            redirect_uri=REDIRECT_URI
        )
        # Создаем редирект с токеном в заголовке (через установку cookie)
        response = RedirectResponse(url="/")
        access_token = token_response.get('access_token')
        # Устанавливаем токен в cookie (альтернатива Authorization header для браузера)
        response = append_cookie(response, "access_token", access_token)
        refresh_token = token_response.get('refresh_token')
        response = append_cookie(response, "refresh_token", refresh_token)
        return response

    except Exception as e:
        return HTMLResponse(content=f"<h1>Ошибка авторизации: {str(e)}</h1>", status_code=400)

@app.get("/logout")
async def logout(request: Request):
    """Выход из системы"""
    refresh_token = request.cookies.get("refresh_token")
    if refresh_token:
        try:
            keycloak_openid.logout(refresh_token)
        except Exception as e:
            print(f"Keycloak logout error: {e}")
    response = RedirectResponse(url="/")
    response.delete_cookie("access_token")
    response.delete_cookie("refresh_token")
    return response

if __name__ == '__main__':
    run(app="02_withroles:app", host='0.0.0.0', port=8100, workers=4, log_level='warning')
```